Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. VI (2011), No. 4 (December), pp. 761-778

A Multi-Agent System Architecture for Coordination of the Real-Time Control Functions in Complex Industrial Systems

J. Wu, R. Tzoneva

Jiang Wu, Raynitchka Tzoneva
Cape Peninsula University of Technology
South Africa, 7350 Bellville, Symphony way
Email: wolfgang.jw@gmail.com, tzonevar@cput.ac.za

Abstract: Multi-agent system architecture for coordination of the real-time control functions in complex industrial systems is presented. The problem which must be solved out is how efficiently to organize the interactions between tasks in order to satisfy the functionality and the time restriction of the system. In order to solve this problem, the treatment of the task interactions is separated from the tasks and is implemented by the proposed multi-agent system. A general three level multi-agent system is introduced to manage the interactions and schedule of tasks. A framework of building of the schedule of the tasks is also presented. Finally, the benefits of the proposed architecture are discussed.
Keywords: Middleware, Multi-agent system architecture, Real-time task, Coordination.

1 Introduction

Real-time systems are usually more complex than other systems because they are based on concurrency, have to deal with multiple independent streams of input events and to produce multiple outputs [1]. Real-time systems are not means to get the results as soon as possible, the requirements towards their operation is to get the results at a prescribed point of time within defined time tolerances [2]. Real-time systems are divided into two classes on the basis of their time tolerances (see [3] and [4]): hard real-time systems and soft real-time systems. Hard real-time systems are those whose deadlines for implementation of the tasks must absolutely be met or the system will be considered to have failed (and this failure might be catastrophic), while soft real-time systems allow for some deadlines, at least occasionally, to be missed with only a degradation in performance but not a complete failure. On the basis of the said above it can be concluded that not only the functions of the systems must be completed, but also the time restriction has to be achieved in order to obtain the goals of the real-time system. The functions of the real-time system are represented by different software tasks having a lot of interactions between themselves. In order to achieve the time requirements of the real-time system a schedule of system's tasks and coordination of their interactions must be organized efficiently.

It is supposed in the paper that every real-time function is realized by a task and each task is coded in one individual software program. This means the interactions between the functions or tasks only may be implemented via ports which are virtual/logical data connections that can be used by the programs to exchange data directly, instead of going through a file or other temporary storage location [5].

Agent theory is applied to build a multi-agent system which coordinates the execution of real-time tasks efficiently using the following capabilities of the agents: autonomy [6], social relations [7], reactivity [8] and pro-activeness [9]. The presented in the paper multi-agent system architecture consists of three agents which represent three levels of processes for interactions and schedule manipulation. The interaction agent, the lowest agent, is used to receive requisitions

for interactions from the tasks; the coordinating agent, the middle level agent, is applied to do the coordination of the tasks; the mapping agent, the highest level agent, takes charge of storing the information of how to deal with the interactions of tasks.

The development and running of tasks which realize the functionality of the real-time system are simplified by adopting this multi-agent architecture. The tasks may only focus on how to achieve the aim of the functionality and how to send and receive the interaction information from the multi-agent system. All interactions are treated by the multi-agent system.

2 The Problem of Real-Time Control Functions

The real-time system is a system in which not only functionality but also the time restriction for implementation of these functions (tasks) must be satisfied. Interaction packets between tasks and the order of execution of these tasks are two main factors which impact the period of time for completion of a given function of the system.

Traditionally, the interaction packets between two tasks are implemented directly and it is not necessary to be managed. Thus, task Ta must know how to connect to task Tb and what is the format of the message which Tb has to receive if Ta wants to send a message to Tb. This obviously increases the burden of the engineering work during the development, running and maintenance of the system.

Usually the order of execution of tasks is scheduled by a high level task which calls some type of scheduler. The scheduler evaluates the time restriction of each task and assigns a priority to the task by a specific arithmetic, and then the task possesses the computation resource (CPU and Memory) ordered according to this priority. Different strategies of scheduling have different arithmetic, but all of them consider that the task is the smallest unit to be processed. The process of determination of priority of the tasks is also necessary to be dynamic one because the environment of the real-time system is open and dynamic one.

Some methods such as Giotto [10], LET [11], TDL (see [12] and [13]) and HTL [14] try to solve the problem for scheduling by division of the task into several subtasks which have no internal interaction. These methods dodge the problem that the priority of tasks should be dynamic but this is still no an ideal solution for the problem for interaction packets management.

3 Method for Coordination of the Real-Time Control Functions

To implement real-time system efficiently, the interaction packets between tasks must be managed and the scheduling of tasks must be dynamic. The method provides a Middleware to supply the function of the interaction packet management and the dynamic scheduling control. The method is not supposed to stop a task which is executed, it schedules tasks by controlling the time which invokes and continues execution of the tasks.

3.1 Time points of the task

There is no concurrency within a task, but the tasks in the real-time structure could be executed in parallel [1]. The tasks often execute asynchronously and are relatively independent of each other for significant periods of time. The tasks need to communicate and synchronize their operations with each other from time to time. Therefore, the tasks need to be paused some times to wait for the feedback of communications and synchronization. A task is also possible to be paused by another task which has higher priority and to be re-executed again after the higher priority task releases the computation resource.

Start time point and end time point are essential time points for every task. A task is invoked with parameters and then executed, see Figure 1. That means the necessary parameters are prepared before the start time point. A task releases the computation resource which is used for execution after it completes the execution and output results of the execution at the end time point.

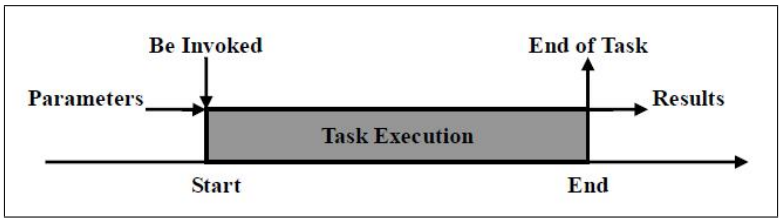

Figure 1: Task with Start and End Time Point of Execution

Almost all of the interaction packets between the tasks (exclude Loosely Coupled Message Communication) make task to pause and continue its execution when the condition is satisfied, see Figure 2. The pause can happen zero or several times during the execution of the task.

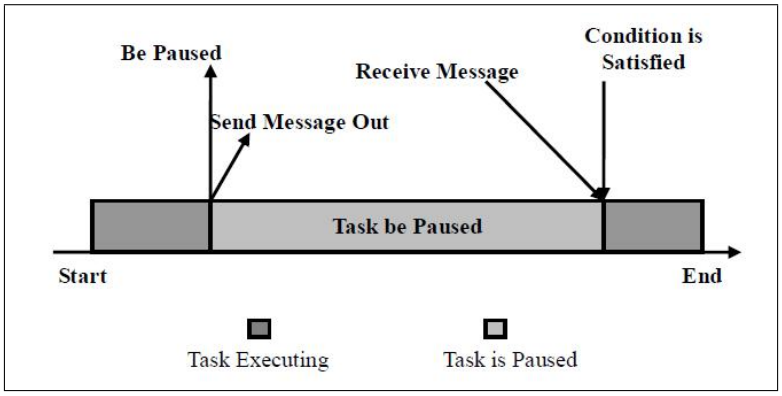

Figure 2: Task to be paused

Loosely Coupled (Asynchronous) Message Communication is a special interaction of the task, it only sends a message out and does not need to wait for any feedback from the other task, see Figure 3. As well as other interactions, it can happen zero or several times during the execution of the task. This point could be called IwF Point (Interaction without Feedback Point).

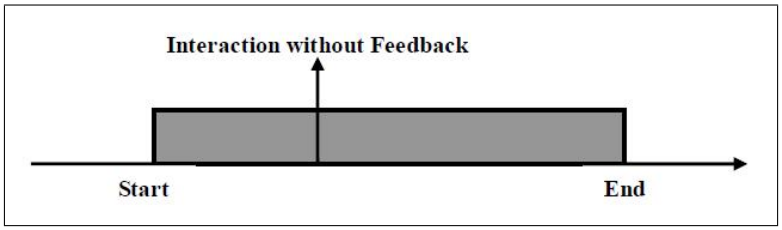

Figure 3: Task's Interaction without Feedback

As mention above, the time points of a task could be Start Point, End Point, Paused Point, Continue Point and IwF Point. Start Point and End Point happen one and only one time during a completed execution of task. Other three time points may happen zero or several times during a completed execution of a given task, and must be noted that Paused Point and Continue Point always happened in couples.

Thus, a task may have five type points which determine the run-able situation of the execution, they are start point and end point, paused point and continue point, and IwF point. The IwF point does not affect the execution of the task. Before start point and after end point, as well as for the time from paused to re-executed points, the task is not able to execute, and the computation resources (at least CPU) are able to be assigned to other tasks.

3.2 Separation of the interactions between real-time tasks

The traditional way of interaction is changed in the new method, the interaction does not directly happen between two tasks in which one is sender and other is receiver. A requisition which contains the information of an interaction packet is sent to the Middleware and then this information is sent to the target task from the Middleware after the requisition is treated by the Middleware. Thus, operation of an interaction packet from task Ta to task Tb is divided into three phases which are: interaction packet sending (Ta to Middleware), interaction packet treatment (Middleware), and interaction packet receiving (Middleware to Tb).

To make the execution of the real-time system more efficient, all possible types of interaction must be registered in the Middleware. And then, the task which sends an interaction packet does not need to know which particular task takes charge to receive the interaction packet information, the Middleware will appoint a task which has capability and also is available to receive and treat the interaction packet. This aspect makes the real-time system more flexible during the development, runtime and even maintenance. Engineers don't need to know which task must receive the interaction packet during development, any interaction packet is simply just sent to the Middleware. There are multi choices of tasks which may treat the interaction packet; the Middleware will appoint one or more tasks to treat the interaction packet by evaluating the status of execution of the real-time system at runtime. This aspect also reduces the workload and limits the effect range when a function is added or removed because the tasks which realize other functions don't need to do any modification to fit the change. The only one requirement is to register the interaction which the new task may send and receive.

A task may receive an interaction packet only when it is not started and then be invoked or it is paused and waiting to receive an interaction packet to continue execution. A task does not possess the computation resource (CPU) before the task starts and continues. The Middleware may schedule the tasks by controlling the time at which it sends the interaction packet to the target task. All changes of the status of the tasks must be sent to the Middleware to monitor the status of the real-time system and schedule the tasks.

Figures 4 and 5 compare the treatment of interaction packets of the traditional way and the new method. One interaction packet is separated into two parts, interaction request sending and interaction packet receiving. Interaction 1 in Figure 4 is separated according to the proposed method into interaction 1A and interaction 1B as can be seen in Figure 5; interaction 2 in Figure 4 is separated into interaction 2A and interaction 2B in Figure 5; interaction 3 in Figure 4 is separated into interaction 3A and interaction 3B in Figure 5. All the interactions in the section A and the section B are connected by the Middleware. The target task of section B is selected by the Middleware and the time at which the Middleware sends interaction packet to the target task is decided by the Middleware by evaluating the situation of the real-time system.

3.3 The mechanism of the coordination

The Middleware takes charge to receive interaction packet and send interaction packet to the target task. An interaction packet must be processed by several phases in the Middleware and then could be sent to the target task. There are six phases to treat an interaction packet in the Middleware. In Figure 6, each ellipse which is located in the Middleware represents a phase of

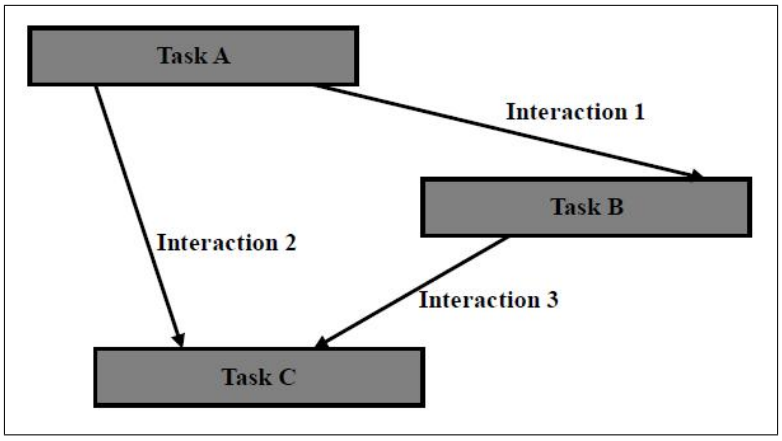

Figure 4: Tasks interact directly

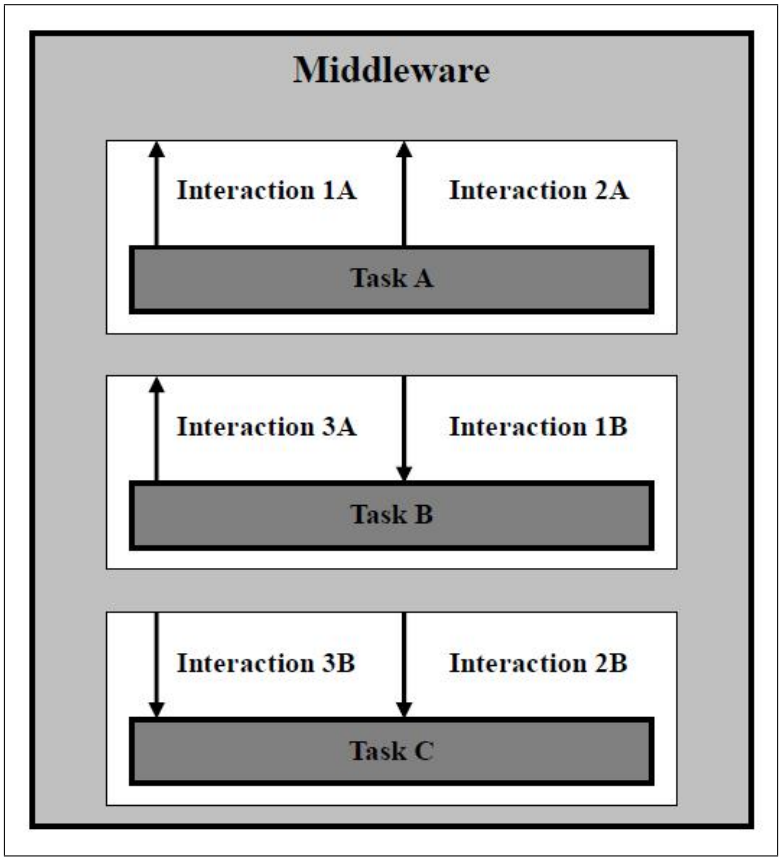

Figure 5: Tasks Interact via Middleware

treatment of interaction packet. By the order of treatment, they are: receive, change status of sending task, find the treatment plan, schedule the tasks, send and change the status of receiving task.

Receiving a requisition of an interaction packet is the first phase. The Middleware must supply a common port to receive the interaction packet. The interaction packet should have several parts. First at all, the sender of the interaction packet must be indicated. The second is the type of the interaction packet. These two parts of the interaction packet is used to find the treatment plan. The third part is the main body of the interaction packet, the message. The last part indicates the time restriction of the interaction packet. This part may not be attached with the interaction packet because some interactions may have no time restriction. After the interaction packet is sent, the task may continue or pause its execution depending on if the interaction needs reply or not. The second phase is changing the status of the task which sends the interaction packet. Any change of the status of tasks must be recorded to monitor the status of the real-time system. The status of the real-time system is foundation for scheduling of the tasks in the Middleware of the real-time system.

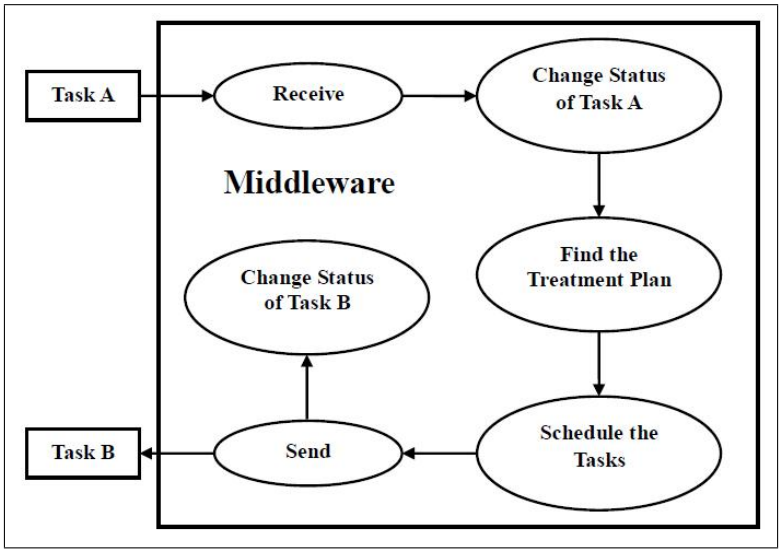

Figure 6: Treatment of Interaction Packet in the Middleware

Finding the treatment plan is the third phase to treat the interaction packet. The Middleware searches the treatment plan of the interaction packet in a database which stores the information of how to treat the interaction packets according to the sender indication and the type of the interaction packet. This search may find several plans, all of these plans must be sent to the next phase. The fourth phase is scheduling the tasks of the real-time system. The Middleware decides which treatment plan will be performed and what time for performing of the plan to be selected based on evaluating the status of the real-time system and the time restriction from the information of the treatment plan and the interaction packet. If the interaction packet is not sent out immediately, it should be stored in a waiting queue, and be sent in time to the target task. The time when the interaction packet will be sent is changeable, every time the Middleware makes the decision affects the waiting queue and the sending time of all the interaction packets which are waiting for sending could be reassigned.

There are two statuses of tasks at which they could receive the interaction packet, one is before the task starts, and other one is the tasks is paused. When a task is executing, it could not be possible to receive any information from outside of the task because the task has no internal concurrency. In addition the interaction packet could contain only the parameters to invoke a

task which is not started or could be a message to make the task to continue the execution. The status of the task which receives the interaction packet is changed when the interaction packet arrives. Thus, the receiving task's status is necessary to be changed at the Middleware for next evaluation of the real-time system. The graphic description of the treatment of an interaction packet is presented in Figure 6. The arrows represent the work flow of treatment of the interaction packet. The Middleware takes charge to find the target task to receive the interaction packet and schedule the tasks by controlling the interaction packet sending time. There are two times where the status of the real-time system changes during treating an interaction packet and reassigning the sending time of the interaction packets waiting in the queue.

4 General Multi-Agent System Architecture for Coordination of the Real-Time Control Functions

The main function of the Middleware is for a specific time to receive the interaction packet from a task, to analyze it, and to send the packet to other tasks which should do further treatment of the packet. To achieve this target, the Middleware should have some information for the system:

* Knowledge of the relationship of the tasks except the tasks of Middleware;
* The time restriction of each interaction;
* A map of the status of tasks which are running except the tasks of Middleware.

The knowledge of the relationship of the tasks is a static global view of the system. It records how to treat an interaction packet and supply the information for further treatment. The time restriction is specified in the relationship of the tasks, it is a guide to indicate when to do the communication.

4.1 Analysis of the Middleware

The objectives of the Middleware are achieved by the cooperation of the three agents.

Objectives of the Middleware

The Middleware does not perform any real function which is required for real-time system from the outside of the system, but it plays an important role in the system to coordinate the tasks which perform the real functions by controlling the communications between them.

The Middleware is also employed to reduce the complexity of the real-time system and makes the system to be developed, maintained and run easily. To achieve this aim, the new method is applied into the Middleware. Therefore, the task which performs some function in the real-time system is required to focus on completing this function only. The task communicates only with the Middleware as follows:

* Sends interactions to the Middleware
* Receives interactions from the Middleware.

The interactions between the tasks are treated by the Middleware to reveal the relationship of the tasks in the system.

Thus, the main function of the Middleware is treating the interactions between the tasks of the real-time system to make the system running correctly and in the logic way. The treatment of the interactions also has to satisfy the time restriction because it is working for a real-time control system. The correctness of the real-time system is based on the correctness of both of these two factors. So, the objectives of the Middleware are:

* Treat the interactions between the tasks to reveal the relationship of the tasks in the system

* Treat the interactions on time to satisfy the time restriction of the real-time system.

Middleware's Agents

It is proposed the Middleware functions to be implemented by three agents: an Interacting Agent, a Coordinating Agent, and a Mapping Agent. Each agent plays a distinct role to complete a specified function. These agents are self-existent, they work together by exchanging information to achieve the objectives of the Middleware. Each agent performs a specified function by itself and cooperates with others by exchange of information.

The Mapping Agent is an agent which supplies a static, global description of the real-time system. The description is expressed by the view of the relationships of the tasks of the system. This view is detailed through the records of the interactions between the tasks of the system. In fact, the rules for the running of the real-time system (for the interactions and their corresponding treatment) are predefined in the Mapping Agent only, and no undefined rules could be executed in the system.

The Coordinating Agent is an agent which supplies a dynamic, real-time status description of the real-time system. The description of the running system is expressed by the view of the status of the tasks. The status of tasks is changed in real-time; it indicates each task's working condition and some details. Usually, once a task starts, the most details are not changed except the working condition. The Coordinating Agent also schedules the tasks by scheduling the sending time of the orders based on the information which is supplied by the Mapping Agent, the view of the status of the tasks of the system and the schedule arithmetic which is predefined. The Coordinating Agent is the core agent in the Middleware. It executes the schedule arithmetic and the rules of the system by obeying the real-time status of the system.

The Interacting Agent is an agent which temporary stores the main body of the interaction information, which is the real content that should be treated by other tasks. The content of the interaction is only sent to the target tasks and no more treatment is done in the Middleware. The content should be kept until it is not useful. The Interacting Agent separates the interaction into the content and the metadata which is information for the interaction. The Interacting Agent also combines the content and the orders from the Coordinating Agent to form a complete order. The receiving and sending of the interactions and the orders are other two important functions of the Interacting Agent. The Interacting Agent is the only agent which communicates directly with the other tasks of the real-time system.

Thus, the Middleware consist of three agents and each agent represents a control level of the Middleware, a rough view of the structure of the Middleware is given in Figure 7.

Environment of the Middleware in the Real-Time System

The Middleware coordinates the tasks by controlling the interactions between the tasks in the real-time system.

In a real-time system which does not employ the Middleware to coordinate the tasks, a task has to spend a lot of time to treat the interactions with other tasks. In this process the developer

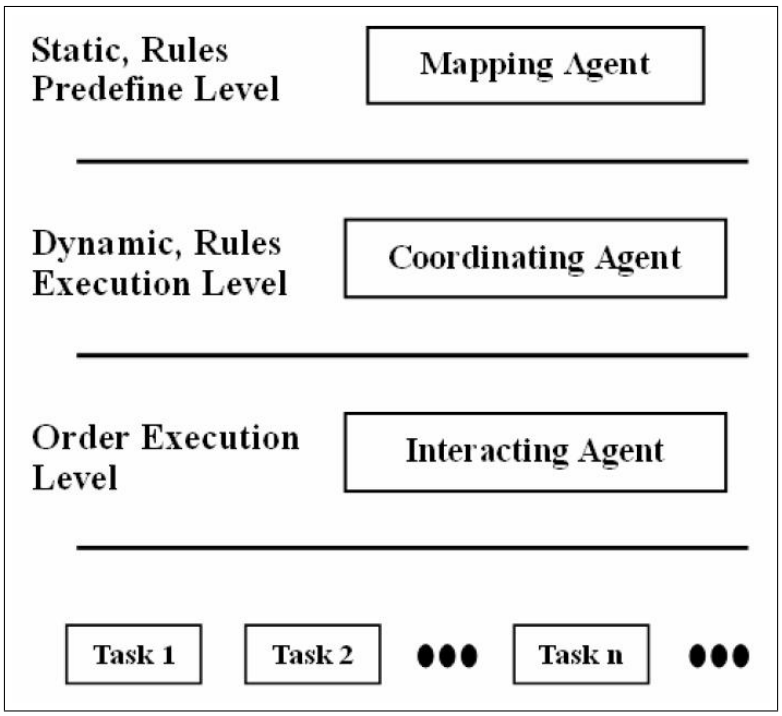

Figure 7: A rough view of the structure of the Middleware

has to know all the details of the tasks which are supposed to be interacted. Additionally various tasks may be forced to do modification when one task is modified. It is also difficult to check the running status of the tasks and difficult to add or remove calculating resources from the system.

The development, maintenance and running of the tasks in the real-time system which employs the Middleware are much simpler than others because they may interact with other tasks by a few steps which are simple and standard. For instance, a sensor task collects value from an electrical sensor. The value then should be sent to a database to be stored and sent to a calculation module to do the further treatment. In a real-time system which does not employ the Middleware, the sensor task should do two interactions with other two tasks to treat one value, and also should have the information of other two tasks. In the real-time system which employs the Middleware, the sensor task should only send the value to the Middleware and only need to have the information of the Middleware. The difference between these two ways of treatment is present in Figure 8.

An interaction usually transfers the information from an initiator (task) to a receiver (task). There are two basic parts of the interaction, one is the content of the information, and other is the order of the information which indicates the receiver how to treat the content of the information. The content of the information could be null when the initiator only wants to inform the receiver that the status of the initiator is changed. The receiver mode is the same in the system with Middleware and in the system without Middleware.

In the system without Middleware, the initiator sends the interactions to the receiver directly and without middle treatment between the initiator and the receiver, so the interaction which is sent by the initiator must exactly be the same as the interaction which is received by the receiver.

In the system with Middleware, the interactions are treated by the Middleware. Therefore, the interaction which is sent from the initiator does not indicate how the interaction will be treated by the receiver, but it indicates which type the interaction is. The type of the interaction

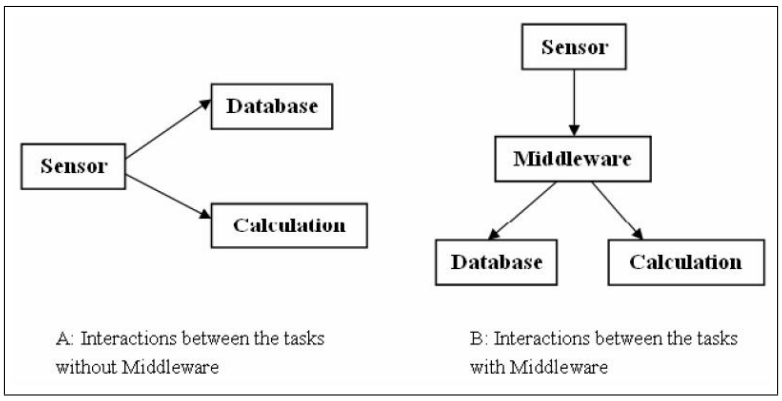

Figure 8: Difference of the interactions between the system with and without Middleware

and the type of the task constitute a unique symbol which could be analyzed in the Middleware. The Middleware may parse the symbol and obtain several orders to the receivers and then send to them these orders with the content of the information.

Figure 9 presents different types of the interactions. The grey area of the two charts are the same, they represent that the orders which the receiver has received are the same. The different is the initiator which only sends one interaction to the Middleware in the system which has Middleware, and the initiator sends two interactions to the receivers in the system without Middleware. Actually, no matter how many tasks receive orders to treat the content of the information, the initiator only needs to send one interaction to the Middleware in the system which is with Middleware. The Middleware parses the task type and the interaction type to obtain the receivers and the orders and then sends the orders to receivers. In the system without Middleware, the initiator has to send separate orders to every receiver by itself. In a complex system, the number of orders is very big, it could be hundreds even thousands orders that have to be sent by the initiator itself.

The workflow of treatment of the interaction

There are three types of interactions, and the different type of interactions should be treated by different steps. The interactions are:

* Task register interaction;

* Task status change interaction;

* The task should communicate with other tasks.

The procedure of treatment of an interaction starts once the Middleware receives the interaction. The whole procedure usually consists of different steps for different type of interactions.

For the first type interaction, the procedure is:

* Assign a task No. to the task;

* Return the task No. to the task;

* Register the task at the Middleware.

For the second type interaction, the procedure is:

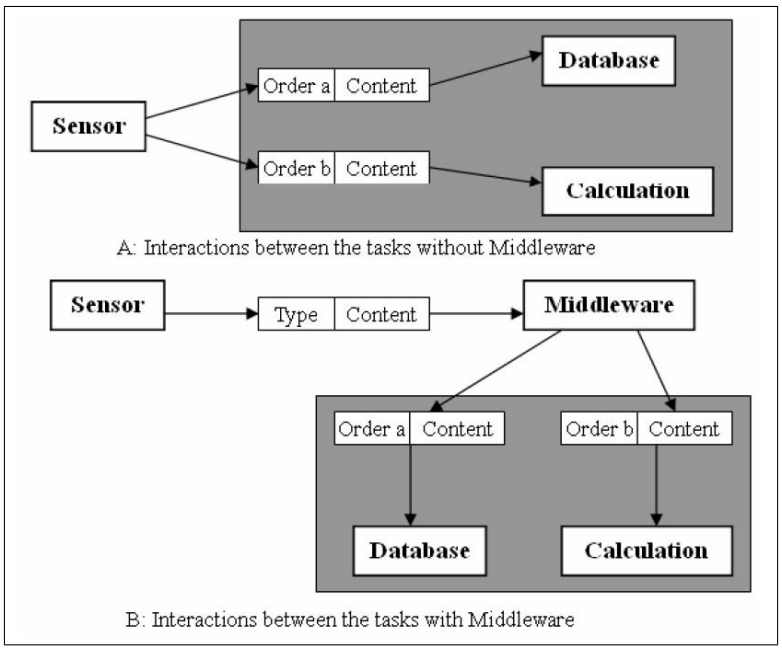

Figure 9: Different type of interaction procedures

* Return a received signal to the task;
* Change the status of the task.

For the third type interaction, the procedure is:

* Return a received signal to the task;
* Separate the metadata from the interaction;
* Change the status of the task;
* Find the order plans for the interaction;
* Select a plan to be executed;
* Combine the order and the content;
* Execute all orders.

The whole workflow of treatment of the interaction is presented in Figure 10.

The interactions between the agents

According the Figure 10, there are two bounds between the agents, they are: the bound between the Mapping Agent and the Coordinating Agent, and the bound between the Coordinating Agent and the Interacting Agent. Each bound has two data transfer directions. In additional of these, the Interacting Agent also has other two data transfer directions to communicate with the outside of the Middleware (receive an interaction and execute the order). Therefore, there are six different data transfers in the Middleware, and they are listed by the order of treatment:

* From outside of the Middleware to the Interacting Agent;

* From the Interacting Agent to the Coordinating Agent;
* From the Coordinating Agent to the Mapping Agent;
* From the Mapping Agent to the Coordinating Agent;
* From the Coordinating Agent to the Interacting Agent;
* From the Interacting Agent to outside of the Middleware

The description below gives the types of data transfer for the third type of the interactions of a task with other tasks.

* In the first data transfer, the intact interaction is transferred from outside of the Middleware to the Interacting Agent. And then the whole interaction is separated into metadata and the content. The content is stored in the Interacting Agent and the metadata is sent to the Coordinating Agent.
* The second data transfer is: the Coordinating Agent receives the metadata from the Interacting Agent. The metadata should indicate the initiator of the interaction, and then the Coordinating Agent is able to change the status of the task which is the initiator.
* The third data transfer is: the Mapping Agent receives the metadata from the Coordinating Agent. The metadata should indicate the unique interaction type, and the interaction type will be used to find the order plans from the database.
* The fourth data transfer is: the Mapping Agent returns the search results which are a set of order plans to the Coordinating Agent. Each plan consists of several orders. The scheduler is able to select a plan to be executed on the bases of the status of the tasks.
* The fifth data transfer is: the Coordinating Agent sends the orders to the Interacting Agent to combine the content which was separated before. All of these orders should be able to obtain the correct content which should be carried to be sent to other tasks.
* The last data transfer is: the Interacting Agent sends the intact orders to the tasks which should receive the orders.

For the other two types of the interactions only the first and the second data transfer are involved.

The data containers in the agents

To supply a static, global description of the real-time system, the Mapping Agent involves a database to store the information of the system. The database should have the information of the task type, its interaction type and how to deal with the interaction. It should be able to return the order plans by supplied task type and the interaction type. To supply a dynamic, real-time status description of the system, all running tasks' status must be stored in a date container and must be changed dynamically. It is the basis for scheduling the tasks. No matter which schedule arithmetic is applied, the real-time status of the system is always useful for the scheduling.

The Interacting Agent keeps the content of the interaction in a data container. The content should be connected to the order must be removed when it will not be used again. The data in the Mapping Agent is static and should only be read. Other two data containers have both

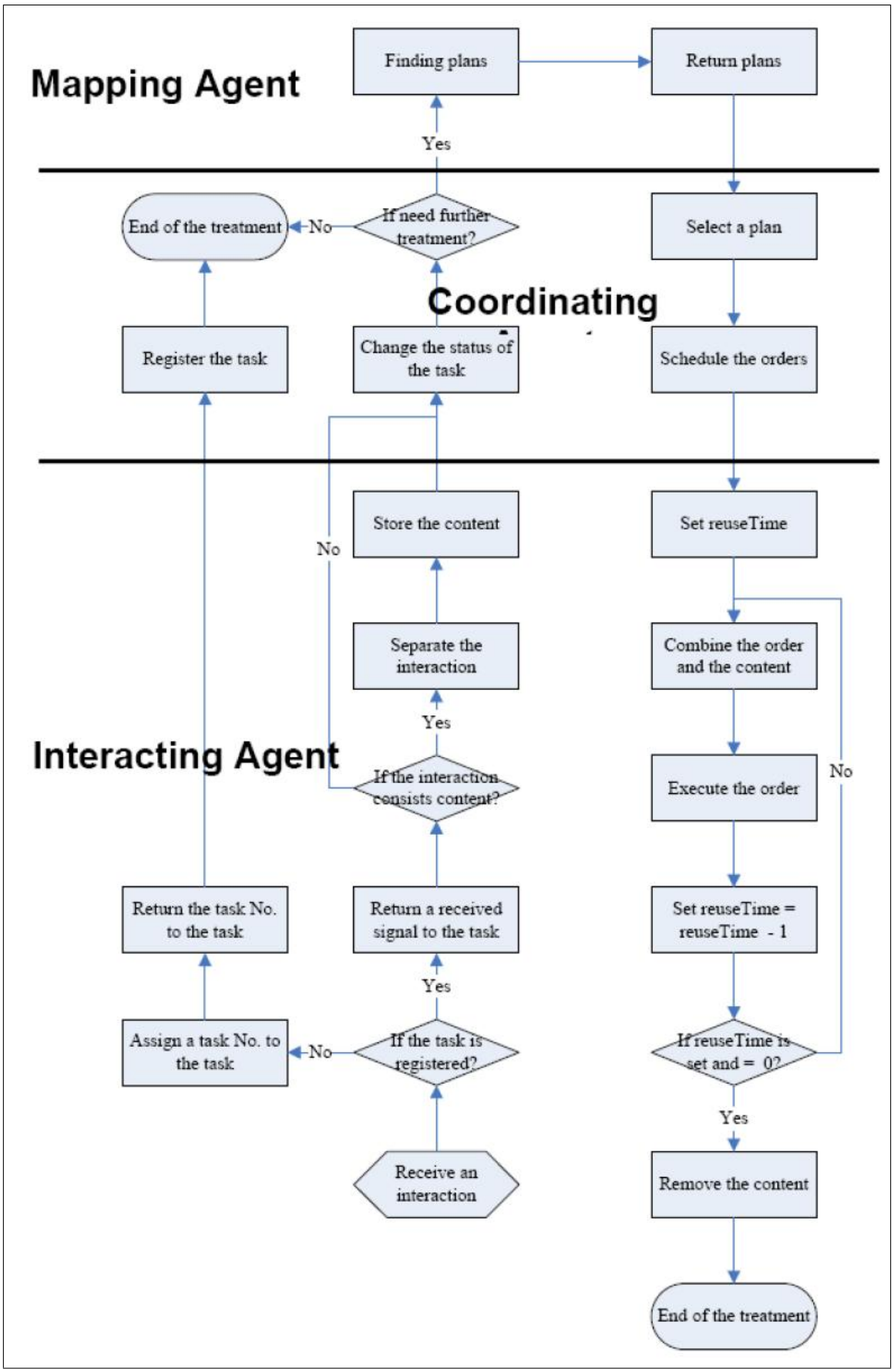

Figure 10: The workflow of treatment of the interaction

read and write (add, remove, change) operations. For each of these, it is possible that two write operation to happen at the same time or a read and a write operation to happen at the same time. This may lead to potential dangerous (unpredictable results) of the data. To avoid this dangerous situation, no other operation may be allowed to be executed when a write operation is happening.

4.2 Architecture of the Middleware

The Middleware is working in a real-time system, and has to communicate with other tasks of the system. Each task is a self-existent program, and they may be distributed in the network and working concurrently. A real-time control system usually is a concurrent system and requests the coordination component to have high response efficiency. Figure 11 presents the detailed architecture of the Middleware.

Data is the core

Three data containers exist to store the information of the system. Each data container is the core of the agent in which it stays. The agents' actions are all around these data containers. The data containers and specific agents are listed below in a Table 1:

Data	Agent
Relationship of the tasks	the Mapping Agent
Running Status	the Coordinating Agent
Content of Interactions	the Interacting Agent

Table 1: Data Containers and Specific Agents

Queues and communicators

The queues and the communicators are the channels of the transfer of the data between the agents.

A queue is a data structure which may store elements and order them in a FIFO (first-in-first-out) manner. The communicators are pair operations on the queue, one is to get the element and other is to put the element. Seven queues exist in the Middleware to transfer data between the agents. They are listed below in a Table 2:

Queue	From (get)	To (put)
qSisiIn	SISI	the Interacting Agent
qInCoor	the Interacting Agent	the Coordinating Agent
qCoorMap	the Coordinating Agent	the Mapping Agent
qMapCoor	the Mapping Agent	the Coordinating Agent
qCoorInA	the Coordinating Agent	the Interacting Agent
qCoorInB	the Coordinating Agent	the Interacting Agent
qInSisi	the Interacting Agent	SISI

Table 2: Queues in the Middleware

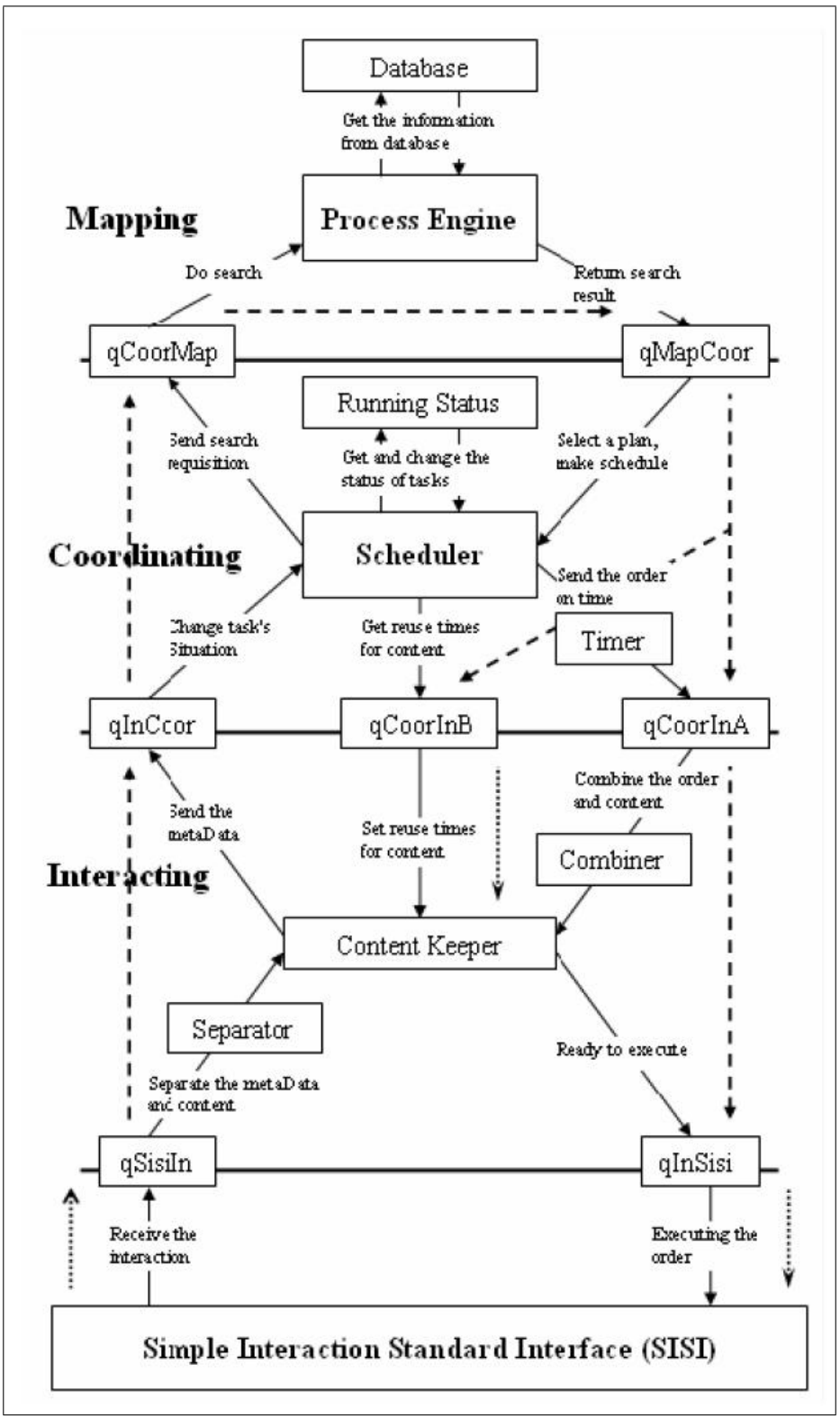

Figure 11: Detailed Architecture of the Middleware

Simple Interaction Standard Interface (SISI)

Simple Interaction Standard Interface (SISI) is a special module of the Interacting agent of the Middleware. It is the only module that communicates with the tasks directly. It receives the interaction packets from the tasks, put them into the qSisiIn and sends the order packets from the qInSisi to the tasks.

Threads

Two types of threads exist. One is running during the lifetime of the Middleware, and the other is started by other thread. The first type of the threads is named "Listen Thread" because they always listen the queues and create a second type threads which are executing the real treatment. So the second type is named "Function Thread".

Listen threads are established when the Middleware starts and they run during the lifetime of the Middleware. Each of them is monitoring a specific queue to check if there is any element in the queue and get the element from the queue by the rule "first-in-first-out". To shorten the time of the response, the listen thread creates a function thread to do the treatment of the element which is gotten from the queue and go back to monitoring the queue immediately. A Process Head which is a special thread listens the port to receive the interaction packet in SISI. It creates a new thread to do the first treatment of the interaction packet and put the result as an element into the queue qSisiIn.

Function threads execute the real treatment of the elements. They usually read or write something from the data container, and also obtain a new element for the further treatment except no further treatment is needed. If the thread obtains a new element which needs further treatment, the function thread then invokes a put function to put the element into the queue which receives the element for next step treatment. A Process End is created by a listen thread QGetInSisi. It sends the order packet to the tasks which should receive the order packet. It is the last function in the treatment cycle.

Timer

Timer is a module in the Coordinating Agent of the Middleware. It executes the specific function in a specific time. The specific function here is put the element into the queue qCoorInA. The element arrives to the queue qCoorInA at the specific time.

The Middleware receives the interactions from the tasks and sends the orders to the tasks. Each agent has a data container to store the information which should be treated. The queues and the listen threads and the put functions provide the capability that the agents may communicate with each other. The real treatments are executed in the threads which are created by listen threads.

5 Scheduling in the Coordinating Agent

Coordinating Agent does real coordination in the Middleware for the whole real-time control system. It schedules tasks in the real-time system and manages the interactions between them. It is placed between the Interacting Agent and the Mapping Agent and consists of a scheduler and running status.

Scheduler is used to organize executions of tasks of the system in the Coordinating Agent. It schedules tasks based on the clock, the status of the system, the meta-data of interaction packets from the Interacting Agent and the sets of treatment plans from the Mapping Agent. Scheduler decides which plan of treatment is used for response to the interaction packet based

on the status of the system. Scheduler also decides at what time to send the order to the Interacting Agent. That is the scheduler decides which order or orders are used to response to the requisition of interaction packet, and at what time the order is sent to which task. Scheduler controls the execution of the system dynamically through control of the execution of the tasks and the interaction packets between the tasks.

Running status is a real-time table to record execution details of the tasks which are running at the time. Any change of the status of the task is recorded in the running status after scheduler receives the meta-data of the interaction packet from the Interacting Agent. Running status supplies the information to the scheduler for scheduling the orders. The order which is sent to the Interacting Agent is also recorded in the running status. Several tasks may run the same code of a program at the same time or a same code of a program may be run several times in the system. A code of the task exists in the running status to indicate which code of the program is run by the task.

6 Conclusions

The Middleware coordinates the tasks in a real-time control system by scheduling them and managing the interaction packets between them. The Middleware could be thought as a post office. The letters (interaction packets) are collected by postman (Interacting Agent) first. Then the post office (Coordinating Agent) sorts the letters based on the yellow page (Mapping Agent). At the end, the postman sends the letter to the addressee.

The whole process of treatment of an interaction packet in the Middleware could be divided into three phases:

* Requisition collection;
* Finding solution;
* Processing of the solution.

Comparing the traditional solutions with the proposed method, the benefit of the method is obvious. First at all, the whole real-time control system is separated into three parts: function realization, schedule execution and system architecture. This makes each part of the system to be developed in more effective and professional way. Each of these three parts is easy to be changed. The table of interaction packets may be changed to realize another purpose. The scheduler may be realized by other type arithmetic for changing the strategy of the scheduling. The tasks may be added, removed or upgraded easily because only some corresponding changes are necessary to be added to the database which is placed in the Mapping Agent.

System status is easy to be monitored during system run, because this status is a snapshot of the system.

Acknowledgements

The investigations are funded by the National Research Foundation of South Africa under the grant 62364 “Substation Automation and Energy Management Systems”.

Bibliography

[1] H. Gomaa, *Designing Concurrent, Distributed, and Real-Time Applications with UML*, Addison-Wesley, 2000.

[2] A. Gambier, Real-time control systems: a tutorial, *The Fifth Asian Control Conference*, Melbourne, Australia, 2004.

[3] C. Liu, J. Layland, Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, *Association for Computing Machinery*. Vol.20, No. 1, pp. 46-61, 1973.

[4] E. Jensen, C.Locke, H. Tokuda, A time-driven scheduling model for real-time operating systems, *Proceedings of the IEEE Real-Time Systems Symposium*, 1985.

[5] http://en.wikipedia.org/wiki/Computer_port_(software)

[6] C. Castelfranchi, Guarantees for autonomy in cognitive agent architecture. In *M. Wooldridge, N. Jennings, (eds), Intelligent Agents: Theories, Architectures, and Languages*. LNAI Vol.890, pp.56–70,Springer-Verlag: Heidelberg, 1995.

[7] M. Genesereth, S. Ketchpel, Software agents. *Communications of the ACM*, 37(7), pp.48–53, 1994.

[8] H. Chebeane, F. Echalier, Towards the use of a multi-agents event based design to improve reactivity of production systems, *Computers & Industrial Engineering*, Vol.37, No.1-2, pp.9-13, 1999.

[9] M. Wooldridge, N. Jennings, Intelligent agents: theory and practice, *The Knowledge Engineering Review*, 10(2), pp.115-152, 1995.

[10] T. Henzinger, B. Horowitz, C. Kirsch, Giotto: A time-triggered language for embedded programing, *Proceedings of the IEEE 91*, pp.84–99, 2003.

[11] T. Henzinger, C. Kirsch, M. Sanvido, W. Pree, From control models to real-time code using giotto, *IEEE Control Systems Magazine* 23(1), p.50–64, 2003.

[12] E. Farcas, C. Farcas, W. Pree, J. Templ, Transparent distribution of real-time components based on logical execution time, *In Y. Paek, R. Gupta (eds.): LCTES, ACM*, pp.31–39, 2005.

[13] C. Farcas, W. Pree, A deterministic infrastructure for real-time distributed systems, In: *OSPERT 2007 Workshop on Operating Systems Platforms for Embedded Real-Time applications*, 2007.

[14] A. Ghosal, T. Henzinger, D. Iercan, C. Kirsch, A. Sangiovanni-Vincentelli, Hierarchical coordination language for interacting real-time tasks, In: *Proceedings of the 6th ACM International Conference on Embedded software, Seoul, Korea*, ACM (Oct 2006), pp.132–141, 2006.